Building and Running Apps on Kestrel

Timothy H. Kaiser, Ph.D.
tkaiser2@nrel.gov

ABSTRACT

This talk will cover the parallel and serial (MPI/C/Fortran) programming and execution environments on Kestrel. We'll approach this from a "toolchain" perspective. That is, well show makefiles which load the required modules then compile parallel (hybrid MPI/OpenMP) and serial Fortran and C codes. Then we'll show the commands to run the applications on compute nodes.

We'll tie this all together by presenting a script that will make and run example codes using several programming environments including PrgEnv-cray, PrgEnv-gnu, PrgEnv-intel, and IntelMPI, OpenMPI.

The final result of running this script will be output from programs for each programming environment.

We'll also discuss nuisances of the PrgEnv-* environments and some surprising interactions between Cray's gcc compiler modules and IntelMPI **and various Intel modules.**

One of the programs will be discussed in more detail and we'll show how it can be used to diagnose affinity issues.

ABSTRACT

The National Renewable Energy Lab has just launched (and submitted a top 500 Run) its newest HPC platform, Kestrel, with 2144 nodes with dual Intel Sapphire Rapids processors. We'll discuss MPI/OpenMP affinity mapping and testing. We'll present a hybrid MPI/OpenMP test code that reports affinity as a function of environmental settings, tasks, and threads. We show that without attentiveness to affinity performance can be adversely effected However, we'll show how to get ideal mapping, where tasks and threads are laid out for performance.   We present a batch script that can be run to sweep over various command line, environmental settings and task/thread combinations. In addition to Intel compilers the script will test Cray, MPICH, and OpenMPI. We recommend the test code be run before a production run to ensure the desired mappings of tasks and threads. A git repository will be available with all codes and scripts.

CURRENT KESTREL CONFIGURATION

Number of Nodes	**Processors**	**Memory**	**Accelerators**	**Local Storage**
2144	Dual socket **Intel Xeon Sapphire Rapids** (52-core) processors	256 GB DDR5	N/A	256 nodes with 1.92 TB NVMe M.2
10	Dual socket Intel Xeon Sapphire Rapids (52-core) processors	2 TB DDR5	N/A	8 x 1.6 TB NVMe
8	Dual socket Intel Xeon Sapphire Rapids (52-core) processors	256 GB DDR5	2 NVIDIA A40 GPUs	2 x 3.84 TB NVMe

SCHEDULE

- Some links
- What's up with all the module paths
- Module tricks
- Compile/Run with Intel compilers and Intel MPI
- A bunch of Intel modules
- Prog-*
    - What are they?
    - What is craype-x86-spr
    - More compiler wrappers
    - Compile/Run with Prog-*
    - Interactions between Intel MPI and Prog-*
- Affinity testing - What's it about and why is it important?
- Complete example script

SOME LINKS

- https://nrel.github.io/HPC/Documentation/
- https://nrel.github.io/HPC/Documentation/Systems/Kestrel/Environments/
- https://nrel.github.io/HPC/Documentation/Systems/Kestrel/Environments/Toolchains/
    - **Examples:**

```
tar -xzf /nopt/nrel/apps/examples/recreate.tgz
OR
git clone git@github.nrel.gov:hpc-apps/kestrel-tds.git  ;  cd kestrel-tds/affinity/tutorial
sbatch -A MYACCOUNT script
```

- **Near Future: https://github.com/NREL/HPC/tree/master/kestrel**

- https://nrel.github.io/HPC/Documentation/Environment/shell/

MODULE PATHS

```
module avail 2>&1 | grep "\-\-" | sed s/\-//g | sort
 /etc/modulefiles
 /nopt/lmod/modulefiles/core
 /nopt/lmod/modulefiles/mix_compilers
 /nopt/nrel/apps/modules/default/application
 /nopt/nrel/apps/modules/default/compilers_mpi
 /nopt/nrel/apps/modules/default/utilities_libraries
 /opt/cray/modulefiles
 /opt/cray/pe/lmod/modulefiles/core
 /opt/cray/pe/lmod/modulefiles/cpu/x86spr/1.0
 /opt/cray/pe/lmod/modulefiles/craypetargets/default
 /usr/share/lmod/lmod/modulefiles/Core
```

- Any path with nrel in the name was installed by us, contains programs, newer compilers, utilities...
- Everything else is part of the base system
- Any module with mixed in the name is "support" and should not be loaded directly
- Part of the complexity is related to interdependence of modules

Some module "tricks"

```
[tkaiser2@kl1 ~]$module list    # these are the default modules

Currently Loaded Modules:
  1) craype-x86-spr       3) craype-network-ofi       5) cce/15.0.0      7) cray-dsmml/0.2.2    9) cray-libsci/22.12.1.1
  2) libfabric/1.15.2.0   4) perftools-base/22.12.0   6) craype/2.7.19   8) cray-mpich/8.1.23  10) PrgEnv-cray/8.3.3


[tkaiser2@kl1 ~]$module purge
[tkaiser2@kl1 ~]$module list
No modules loaded
[tkaiser2@kl1 ~]$module load openmpi/4.1.5-gcc        gcc/13.1.0
[tkaiser2@kl1 ~]$module list

Currently Loaded Modules:
  1) openmpi/4.1.5-gcc   2) gcc/13.1.0


[tkaiser2@kl1 ~]$module save myopen
Saved current collection of modules to: "myopen"

[tkaiser2@kl1 ~]$module purge
[tkaiser2@kl1 ~]$
[tkaiser2@kl1 ~]$module restore system       # this restores the default modules
Resetting modules to system default. Reseting $MODULEPATH back to system default. All extra directories will be removed from $MODULEPATH.
[tkaiser2@kl1 ~]$module list

Currently Loaded Modules:
  1) craype-x86-spr       3) craype-network-ofi       5) cce/15.0.0      7) cray-dsmml/0.2.2    9) cray-libsci/22.12.1.1
  2) libfabric/1.15.2.0   4) perftools-base/22.12.0   6) craype/2.7.19   8) cray-mpich/8.1.23  10) PrgEnv-cray/8.3.3


[tkaiser2@kl1 ~]$module restore myopen   # this restores the my set of modules
Restoring modules from user's myopen
[tkaiser2@kl1 ~]$module list

Currently Loaded Modules:
  1) openmpi/4.1.5-gcc   2) gcc/13.1.0

[tkaiser2@kl1 ~]$
```

INTEL COMPILERS WITH INTELMPI

- Briefly discuss example programs
- Same compilers are available on Eagle/Swift/Vermilion and are "common"
- Go over a misconception about compiling with mpicc and mpif90
- Compilers generating warnings
- We'll
  - Show modules for builds and runs
  - Build commands
  - Run script extras
  - Example output
  - Makefile

OUR EXAMPLES

- **phostone.c** **fhostone.F90**
  - Hello world on steroids
  - Hybrid MPI / OpenMP
  - Many command line options
  - Will use options to print out a line for each MPI task and OpenMP thread along with the node and core on which it is running
  - Will run for 7 seconds.

PHOSTONE.C OUTPUT

```
MPI VERSION Intel(R) MPI Library 2021.10 for Linux* OS
task    thread             node name  first task    # on node  core
0000      0000       X1005C4S5B0N0         0000         0000  0000
0000      0001       X1005C4S5B0N0         0000         0000  0001
0000      0002       X1005C4S5B0N0         0000         0000  0002
0000      0003       X1005C4S5B0N0         0000         0000  0003
0000      0004       X1005C4S5B0N0         0000         0000  0004
0000      0005       X1005C4S5B0N0         0000         0000  0005
0000      0006       X1005C4S5B0N0         0000         0000  0006
0000      0007       X1005C4S5B0N0         0000         0000  0007

0001      0000       X1005C4S5B0N0         0000         0001  0052
0001      0001       X1005C4S5B0N0         0000         0001  0053
0001      0002       X1005C4S5B0N0         0000         0001  0054
0001      0003       X1005C4S5B0N0         0000         0001  0055
0001      0004       X1005C4S5B0N0         0000         0001  0056
0001      0005       X1005C4S5B0N0         0000         0001  0057
0001      0006       X1005C4S5B0N0         0000         0001  0058
0001      0007       X1005C4S5B0N0         0000         0001  0059

0002      0000       X1005c4s6n0n0         0002         0000  0000
0002      0001       X1005c4s6n0n0         0002         0000  0001
0002      0002       X1005c4s6n0n0         0002         0000  0002
0002      0003       X1005c4s6n0n0         0002         0000  0003
0002      0004       X1005c4s6n0n0         0002         0000  0004
0002      0005       X1005c4s6n0n0         0002         0000  0005
0002      0006       X1005c4s6n0n0         0002         0000  0006
0002      0007       X1005c4s6n0n0         0002         0000  0007

0003      0000       X1005c4s6n0n0         0002         0001  0052
0003      0001       X1005c4s6n0n0         0002         0001  0053
0003      0002       X1005c4s6n0n0         0002         0001  0054
0003      0003       X1005c4s6n0n0         0002         0001  0055
0003      0004       X1005c4s6n0n0         0002         0001  0056
0003      0005       X1005c4s6n0n0         0002         0001  0057
0003      0006       X1005c4s6n0n0         0002         0001  0058
0003      0007       X1005c4s6n0n0         0002         0001  0059
total time      7.001
```

- Intel MPI
- 2 MPI tasks / node
- 8 OpenMP threads
- Sorted and enhanced

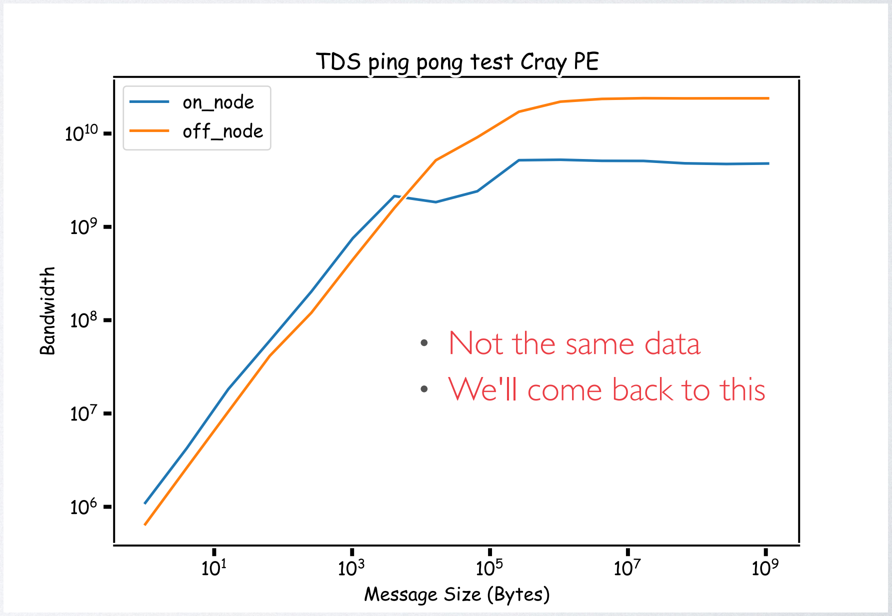
OUR EXAMPLES

- **ppong.c**
  - Measures bandwidth between MPI tasks as a function of message size
  - Measures the MPI_Barrier rate
  - Typically run with N tasks per node where N is the the number of cores
    - By default will run between every set of 2 MPI tasks 104x103 = 10,712 sets
    - The file "todo" gives a subset of tasks to test (2 tasks on first node and 2 on the second)

PPONG.C OUTPUT

```
MPI VERSION Intel(R) MPI Library 2019 Update 10 for Linux* OS

calling MPI_Send - MPI_Recv
1 x1000c1s0b0n0 4.761803e-10 1.392327e-07
2 x1000c1s0b0n1 4.761803e-10 1.725275e-07
3 x1000c1s0b0n1 4.761802e-10 9.615906e-08
 S  R        Size     Min Time     Ave Time     Max Time        Bandwidth    #
 0  1           1 7.519964e-07 1.267607e-06 7.029879e-06       2.6596e+06  200
 0  1           4 7.567462e-07 9.207008e-07 1.448463e-06       1.0572e+07  200
 0  1          16 7.586554e-07 9.269662e-07 1.387508e-06       4.2180e+07  200
 0  1          64 7.942552e-07 1.001559e-06 2.061017e-06       1.6116e+08  200
 0  1         256 8.448493e-07 1.288456e-06 1.935754e-06       6.0603e+08  200
 0  1        1024 1.669955e-06 1.944235e-06 3.213948e-06       1.2264e+09  200
 0  1        4096 3.240118e-06 3.390790e-06 4.044105e-06       2.5283e+09  200
 0  1       16384 8.213287e-06 8.448832e-06 9.251083e-06       3.9896e+09  200
 0  1       65536 3.007229e-05 3.047969e-05 3.297967e-05       4.3586e+09  200
 0  1      262144 5.973231e-05 6.045180e-05 7.064303e-05       8.7773e+09  200
 0  1     1048576 2.803241e-04 2.858262e-04 3.947082e-04       7.4812e+09  200
 0  1     4194304 1.366746e-03 1.457995e-03 1.646080e-03       6.1376e+09  200
 0  1    16777216 5.132919e-03 5.218553e-03 5.361979e-03       6.5371e+09   97
 0  1    67108864 2.027069e-02 2.048386e-02 2.063082e-02       6.6213e+09   26
 0  1   268435456 8.050813e-02 8.093094e-02 8.161121e-02       6.6685e+09    8
 0  1  1073741824 3.237138e-01 3.242401e-01 3.245610e-01       6.6339e+09    3
...
 0  3           1 2.814471e-05 5.013239e-05 4.272152e-03       7.1061e+04  200
 0  3           4 2.833414e-05 2.902550e-05 3.022437e-05       2.8234e+05  200
 0  3          16 2.806301e-05 2.900259e-05 3.033981e-05       1.1403e+06  200
 0  3          64 2.811824e-05 2.896586e-05 3.021923e-05       4.5522e+06  200
 0  3         256 3.016659e-05 3.136935e-05 3.362375e-05       1.6972e+07  200
 0  3        1024 3.418040e-05 3.728393e-05 4.138465e-05       5.9917e+07  200
 0  3        4096 3.088359e-05 3.221709e-05 3.544362e-05       2.6525e+08  200
 0  3       16384 3.776823e-05 3.949602e-05 4.517029e-05       8.6761e+08  200
 0  3       65536 8.354518e-05 8.565556e-05 8.813986e-05       1.5689e+09  200
 0  3      262144 1.879028e-04 1.898664e-04 1.989481e-04       2.7902e+09  200
 0  3     1048576 6.035536e-04 6.115581e-04 6.173814e-04       3.4747e+09  200
 0  3     4194304 2.284192e-03 2.295765e-03 2.308413e-03       3.6725e+09  200
 0  3    16777216 8.828382e-03 8.882622e-03 9.165776e-03       3.8007e+09   58
 0  3    67108864 3.623108e-02 3.648658e-02 3.770787e-02       3.7045e+09   15
 0  3   268435456 1.448475e-01 1.449654e-01 1.450861e-01       3.7065e+09    5
 0  3  1073741824 5.784841e-01 5.787637e-01 5.790432e-01       3.7123e+09    2
...
Barriers/Second 45165.9
...
```

- Intel MPI
- 2 MPI tasks / node
- Only showing info for two task pairs (0-1 & 0-3)



- Not the same data
- We'll come back to this

INTEL COMPILERS WITH INTELMPI

- Go over a misconception about compiling with mpicc and mpif90
- Compilers generating warnings

mpicc mpiicc mpif90 mpiifort

- mpicc - compile with gcc and Intel MPI
- mpiicc - compile with Intel icc and Intel MPI
- mpif90 - compile with gfortran and Intel MPI
- mpiifort - compile with Intel fortran and Intel MPI
- You can "force" compiles by other backends by setting I_MPI_{CC,CXX,FC,F77,F90}

WHAT ARE: ICX AND IFX?

You may see a warning when compiling with Intel compilers

```
The Intel(R) Compiler Classic Compiler is deprecated...
```

The Intel® oneAPI product packages provide two Fortran compilers. Intel Fortran Compiler Classic (ifort) provides best-in-class Fortran language features and performance for CPU. The Intel Fortran Compiler (ifx) enables developers needing OpenMP* offload to Intel GPUs. The OpenMP* 5.0, 5.1 GPU offload features in ifx are not available in ifort. For now ifort continues to be our best-in-class Fortran compiler for customers not needing GPU offload support.

- Notes:
  - Same for icc and icx except icc is being **replaced** by icx
  - Offload only works directly for Intel GPUs not Nvidia
    (Should be a way to get this two work at least for C++)
  - Warning can be suppressed by export -diag-disable=10441

MODULE LOADS

For Builds

```
module load  intel-oneapi-compilers
module load intel-oneapi-mpi
module load gcc/13.1.0
```

- The module load gcc is optional
- Gives you an newer version of gcc
- Can also load gcc/10.1.0
- Don't load other versions of gcc after loading the Intel modules (More on this later.)

For Running

```
module purge
module load libfabric
```

- Usually don't need to load Intel modules
- libfabric gives access to the network

OUR BUILDS

1. Fortran with: Intel MPI and Intel Fortran compiler
2. C with: Intel MPI and Intel C compiler, older compiler (icc)
3. C with: Intel MPI and Intel C compiler, newer compiler (icx)
4. Fortran with: Intel MPI with gfortran Fortran compiler
5. C with: Intel MPI with gcc C compiler

OUR BUILDS

1. Fortran with: Intel MPI and Intel Fortran compiler

```
mpiifort -O3 -g -fopenmp  ex1.f90
```

2. C with: Intel MPI and Intel C compiler, older compiler (icc)

```
mpiicc -O3 -g -fopenmp  ex1.c
```

3. C with: Intel MPI and Intel C compiler, newer compiler (icx)

```
export I_MPI_CC=icx
mpiicc -O3 -g -fopenmp  ex1.c
```

4. Fortran with: Intel MPI with gfortran Fortran compiler

```
mpif90 -O3 -g -fopenmp  ex1.f90
```

5. C with: Intel MPI with gcc C compiler

```
mpicc -O3 -g -fopenmp  ex1.c
```

MAKE

- Our example at https://github.com/NREL/HPC/blob/master/kestrel/Toolchains/Code/Makefiles/Intel/makefile builds for both Intel and gnu backends
- Here we just do Intel backends
- Or makefile uses a trick
    - Its default target is "recurse"
    - Making "recurse" loads module then calls make again for the actual targets

makefile

Intel MPI

```
SHELL:=/usr/bin/bash

recurse:
	module purge                          ; \
	module load intel-oneapi              ; \
	module load intel-oneapi-mpi          ; \
	module load gcc/13.1.0                ; \
	$(MAKE) -f $(firstword $(MAKEFILE_LIST)) both

both: f.impi c.impi pp.impi

#defines USEFAST
include makefile.include

ifeq ($(USEFAST),yes)
OPS=-DUSEFAST
EXTRA=getcore.o
endif

F90=mpiifort
CC=mpiicc

f.impi: fhostone.F90 $(EXTRA)
	$(F90) $(OPS) $(EXTRA) -fopenmp fhostone.F90 -O3 -o f.impi
	rm -f getcore.o

c.impi: phostone.c
	$(CC) $(OPS) -fopenmp phostone.c -O3 -o c.impi

pp.impi: ppong.c
	$(CC) $(OPS) $(WES) ppong.c -O3 -o pp.impi

clean:
	rm -rf *o *mod* f.impi c.impi pp.impi
```

RUN COMMANDS

- Since phostone and fhostone are hybrid MPI/OpenMP codes we will set OpenMP variables.
    - # threads
    - # thread binding
- We'll also add some options to the srun line ensure we are getting good mapping of tasks and threads to cores
- We can run with various numbers of MPI tasks per node and number of OpenMP threads per task but the two multiplied together should not exceed then number of cores on a node (104).

Our script sets these openmp related variables. The first is familiar. KMP_AFFINITY is unique to Intel compilers. In this case we are telling the OS to scatter (spread) out our threads. OMP_PROC_BIND=spread does the same thing but it is not unique to Intel compilers. So in this case KMP_AFFINITY is actually redundent.

```
    export OMP_NUM_THREADS=3
    ~~export KMP_AFFINITY=scatter~~
    export OMP_PROC_BIND=spread
```

The next line

```
    export BIND="--cpu-bind=v,cores"
```

is not technically used as an environmental variable but it will be used to create the srun command line (In later versions of the script). Passing --cpu-bind=v to srun will casue it to report threading information. The "cores" option tells srun to "Automatically generate masks binding tasks to cores." There are many other binding options as described in the srun man page. This setting works well for many programs.

Our srun command line options for 2 tasks per node and 3 threads per task are:

```
    --mpi=pmi2 --cpu-bind=v,cores --threads-per-core=1 --tasks-per-node=2 --cpus-per-task=3
```

- --mpi=pmi2 : tells srun to use a particular launcher
- --cpu-bind=v,cores : discussed above
- --threads-per-core=1 : don't allow multiple threads to run on the same core. Without this option it is possible for multiple threads to end up on the same core, decreasing performance.
- --cpus-per-task=3 : The cpus-per-task should always be equal to OMP_NUM_THREADS.

SIMPLE SBATCH SCRIPT

```
#!/usr/bin/bash
#SBATCH --job-name="impi"
#SBATCH --nodes=2
#SBATCH --exclusive
#SBATCH --export=ALL
#SBATCH --time=00:10:00
#SBATCH --partition=standard

make -f makeimpi

export OMP_NUM_THREADS=3
export OMP_PROC_BIND=spread

CLA="-i -F -E -t 7"
# -i   : Print MPI_Init times for each task at end of run.
# -F   : Add columns to tell first MPI task on a node and and
#        the numbering of tasks on a node.
# -E   : Print thread info at 'E'nd of the run
# -t 7 : Run for 7 seconds

srun --mpi=pmi2 --cpu-bind=v,cores --threads-per-core=1 --tasks-per-node=2 --cpus-per-task=$OMP_NUM_THREADS  ./f.impi $CLA > f.out
srun --mpi=pmi2 --cpu-bind=v,cores --threads-per-core=1 --tasks-per-node=2 --cpus-per-task=$OMP_NUM_THREADS  ./c.impi $CLA > c.out
```

FORTRAN - OUTPUT

```
[tkaiser2@kl1 src]$cat f.out
MPI Version:Intel(R) MPI Library 2021.10 for Linux* OS

task    thread             node name  first task    # on node  core
total time       7.00
0000      0000         X1006C0S0B1N1        0000         0000   000
0000      0001         X1006C0S0B1N1        0000         0000   001
0000      0002         X1006C0S0B1N1        0000         0000   002
0001      0000         X1006C0S0B1N1        0000         0001   052
0001      0001         X1006C0S0B1N1        0000         0001   053
0001      0002         X1006C0S0B1N1        0000         0001   054
0002      0000         X1006C0S1B0N0        0002         0000   000
0002      0002         X1006C0S1B0N0        0002         0000   002
0002      0001         X1006C0S1B0N0        0002         0000   001
0003      0000         X1006C0S1B0N0        0002         0001   052
0003      0001         X1006C0S1B0N0        0002         0001   053
0003      0002         X1006C0S1B0N0        0002         0001   054
mpi_init    0     1696723552.3310     1696723552.9667      0.6357
mpi_init    1     1696723552.3310     1696723552.9666      0.6356
mpi_init    2     1696723552.3838     1696723552.9746      0.5908
mpi_init    3     1696723552.3838     1696723552.9746      0.5908
[tkaiser2@kl1 src]$
```

C - OUTPUT

```
[tkaiser2@kl1 src]$cat c.out
MPI VERSION Intel(R) MPI Library 2021.10 for Linux* OS

task    thread             node name  first task    # on node  core
total time      7.010
0000      0000         X1006C0S0B1N1        0000         0000  0000
0000      0002         X1006C0S0B1N1        0000         0000  0002
0000      0001         X1006C0S0B1N1        0000         0000  0001
0001      0000         X1006C0S0B1N1        0000         0001  0052
0001      0001         X1006C0S0B1N1        0000         0001  0053
0001      0002         X1006C0S0B1N1        0000         0001  0054
0002      0000         X1006C0S1B0N0        0002         0000  0000
0002      0001         X1006C0S1B0N0        0002         0000  0001
0002      0002         X1006C0S1B0N0        0002         0000  0002
0003      0001         X1006C0S1B0N0        0002         0001  0053
0003      0002         X1006C0S1B0N0        0002         0001  0054
0003      0000         X1006C0S1B0N0        0002         0001  0052
mpi_init 0 1696723560.2818 1696723560.8412       0.5594
mpi_init 1 1696723560.2824 1696723560.8408       0.5584
mpi_init 2 1696723560.2902 1696723560.8409       0.5507
mpi_init 3 1696723560.2902 1696723560.8409       0.5508
[tkaiser2@kl1 src]$
```

PrgEnv-*

PrgEnv-amd/8.3.3
PrgEnv-aocc/8.3.3
PrgEnv-cray-amd/8.3.3
PrgEnv-cray/8.3.3
PrgEnv-gnu-amd/8.3.3
PrgEnv-gnu/8.3.3
PrgEnv-intel/8.3.3
PrgEnv-nvhpc/8.3.3
PrgEnv-nvidia/8.3.3

- Modules for using Cray's MPI with various backend compilers
- Red one currently work (cray, gnu, intel)
- Blue ones are for AMD processors and Nvidia GPUs (coming)
- PrgEnv-cray/8.3.3 is the default
- **All of these use the same MPI Library (Cray-MPICH)**

PrgEnv-cray loads several other modules

```
[tkaiser2@kl1 src]$module purge
[tkaiser2@kl1 src]$module load PrgEnv-cray/8.3.3
[tkaiser2@kl1 src]$module load craype-x86-spr
[tkaiser2@kl1 src]$module list

Currently Loaded Modules:
  1) cce/15.0.0      3) cray-dsmml/0.2.2     5) craype-network-ofi   7) cray-libsci/22.12.1.1   9) craype-x86-spr
  2) craype/2.7.19   4) libfabric/1.15.2.0   6) cray-mpich/8.1.23    8) PrgEnv-cray/8.3.3
```

These are the default modules, including crape-x86-spr

What is craype-x86-spr?

- x86-spr stands for Intel Sapphire Rapids - processors we have on Kestrel
- Loading this module sets environmental variables to allow certain optimizations for the Sapphire Rapids processors
- There are other choices that will not be useful until we get our GPU nodes

NEW COMPILER WRAPPERS

- Most MPI compilers mpicc, mpif90... are more or less wrapper scripts that call the underlying C for Fortran compilers with settings to point to the libraries
- ProgEnv-* takes this on step further
  - ftn = call underlying Fortran compiler and "auto detect" if it is a MPI program and build as such
  - cc = call underlying C compiler and "auto detect" if it is a MPI program and build as such
  - CC = call underlying C++ compiler and "auto detect" if it is a MPI program and build as such

PrgEnv-* wrappers

PrgEnv-*	Fortran (ftn)	C (cc)	MPI
PrgEnv-cray	Cray fortran (ftn)	cc - Clang based	Cray MPICH
PrgEnv-gnu	gfortran	gcc	Cray MPICH
PrgEnv-intel	ifort	icc	Cray MPICH

makefile

PrgEnv-cray

```
SHELL:=/usr/bin/bash

recurse:
  module purge                      ; \
  module load craype-x86-spr        ; \
  module load PrgEnv-cray           ; \
  $(MAKE) -f $(firstword $(MAKEFILE_LIST)) both

both: f.cray c.cray pp.cray

#defines USEFAST
include makefile.include

ifeq ($(USEFAST),yes)
OPS=-DUSEFAST
EXTRA=getcore.o
endif

F90=ftn
CC=cc


f.cray: fhostone.F90 $(EXTRA)
  $(F90) $(OPS) $(EXTRA) -fopenmp fhostone.F90 -O3 -o f.cray
  rm -f getcore.o

c.cray: phostone.c
  $(CC) $(OPS) -fopenmp phostone.c -O3 -o c.cray

pp.cray: ppong.c
  $(CC) $(OPS) $(WES) ppong.c -O3 -o pp.cray


clean:
  rm -rf *o *mod* f.cray c.cray pp.cray
```

Others are the nearly the same

```
tkaiser2-37907s:affinity tkaiser2$ diff makeprgcray makeprgintel
6c6,8
<  module load PrgEnv-cray           ; \
---
>  module load intel                 ; \
>  module load PrgEnv-intel          ; \
9c11
< both: f.cray c.cray pp.cray
---
> both: f.intel c.intel pp.intel
21a223,25
> f.intel: fhostone.F90 $(EXTRA)
>  $(F90) $(OPS) $(EXTRA) -fopenmp fhostone.F90 -O3 -o f.intel
>  rm -f getcore.o
23,25c27,28
< f.cray: fhostone.F90 $(EXTRA)
<  $(F90) $(OPS) $(EXTRA) -fopenmp fhostone.F90 -O3 -o f.cray
<  rm -f getcore.o
---
> c.intel: phostone.c
>  $(CC) $(OPS) -fopenmp phostone.c -O3 -o c.intel
27,28c30,31
< c.cray: phostone.c
<  $(CC) $(OPS) -fopenmp phostone.c -O3 -o c.cray
---
> pp.intel: ppong.c
>  $(CC) $(OPS) $(WES) ppong.c -O3 -o pp.intel
30,33d32
< pp.cray: ppong.c
<  $(CC) $(OPS) $(WES) ppong.c -O3 -o pp.cray
<
<
35c34
<  rm -rf *o *mod* f.cray c.cray pp.cray
---
>  rm -rf *o *mod* f.intel c.intel pp.intel
```

- We load different modules
- Compiler dependent options can change

makefile

PrgEnv-gnu

```
SHELL:=/usr/bin/bash

recurse:
  module purge                ; \
  module load craype-x86-spr  ; \
  module load PrgEnv-gnu      ; \
  module load gcc
  $(MAKE) -f $(firstword $(MAKEFILE_LIST)) both

both: f.gnu c.gnu pp.gnu

#defines USEFAST
include makefile.include

ifeq ($(USEFAST),yes)
OPS=-DUSEFAST
EXTRA=getcore.o
endif

F90=ftn
CC=cc


f.gnu: fhostone.F90 $(EXTRA)
  $(F90) $(OPS) $(EXTRA) -fopenmp  fhostone.F90 -O3 -o f.gnu
# $(F90) $(OPS) $(EXTRA) -fopenmp -fallow-argument-mismatch  fhostone.F90 -O3 -o f.gnu
  rm -f getcore.o

c.gnu: phostone.c
  $(CC) $(OPS) -fopenmp phostone.c -O3 -o c.gnu

pp.gnu: ppong.c
  $(CC) $(OPS) $(WES) ppong.c -O3 -o pp.gnu

clean:
  rm -rf *o *mod* f.gnu c.gnu pp.gnu
```

makefile

PrgEnv-intel

```
SHELL:=/usr/bin/bash

recurse:
  module purge                      ; \
  module load craype-x86-spr        ; \
  module load intel                 ; \
  module load PrgEnv-intel          ; \
  $(MAKE) -f $(firstword $(MAKEFILE_LIST)) both

both: f.intel c.intel pp.intel

#defines USEFAST
include makefile.include

ifeq ($(USEFAST),yes)
OPS=-DUSEFAST
EXTRA=getcore.o
endif

F90=ftn
CC=cc

f.intel: fhostone.F90 $(EXTRA)
  $(F90) $(OPS) $(EXTRA) -fopenmp fhostone.F90 -O3 -o f.intel
  rm -f getcore.o

c.intel: phostone.c
  $(CC) $(OPS) -fopenmp phostone.c -O3 -o c.intel

pp.intel: ppong.c
  $(CC) $(OPS) $(WES) ppong.c -O3 -o pp.intel

clean:
  rm -rf *o *mod* f.intel c.intel pp.intel
```

PrgEnv-* modules

PrgEnv-*	Loads for make	Loads of run (may not be required)

PrgEnv-cray	`module load craype-x86-spr`
	`module load PrgEnv-cray`
		`module load craype-x86-spr`
		`module load PrgEnv-cray`

PrgEnv-gnu	`module load craype-x86-spr`
	`module load PrgEnv-gnu`
	`module load gcc`
		`module load craype-x86-spr`
		`module load PrgEnv-gnu`
		`module load gcc`

PrgEnv-intel	`module load craype-x86-spr`
	`module load intel`
	`module load PrgEnv-intel`
		`module load craype-x86-spr`
		`module load intel`
		`module load PrgEnv-intel`

Example Output PrgEnv-cray

```
MPI VERSION MPI VERSION    : CRAY MPICH version 8.1.23.5 (ANL base 3.4a2)
MPI BUILD INFO : Tue Nov 29 12:42 2022 (git hash 210ae8b)

task    thread             node name  first task    # on node  core
total time      7.004
0000      0000         X1005C2S3B1N1        0000          0000  0000
0000      0005         X1005C2S3B1N1        0000          0000  0005
0000      0007         X1005C2S3B1N1        0000          0000  0007
0000      0002         X1005C2S3B1N1        0000          0000  0002
0000      0001         X1005C2S3B1N1        0000          0000  0001
0000      0004         X1005C2S3B1N1        0000          0000  0004
0000      0006         X1005C2S3B1N1        0000          0000  0006
0000      0003         X1005C2S3B1N1        0000          0000  0003
0001      0007         X1005C3S1B0N0        0001          0000  0007
0001      0002         X1005C3S1B0N0        0001          0000  0002
0001      0004         X1005C3S1B0N0        0001          0000  0004
0001      0005         X1005C3S1B0N0        0001          0000  0005
0001      0003         X1005C3S1B0N0        0001          0000  0003
0001      0001         X1005C3S1B0N0        0001          0000  0001
0001      0006         X1005C3S1B0N0        0001          0000  0006
0001      0000         X1005C3S1B0N0        0001          0000  0000
mpi_init 0 1696732937.5528 1696732937.6665       0.1137
mpi_init 1 1696732937.5150 1696732937.6673       0.1523
```

Example Output PrgEnv-intel

```
MPI VERSION MPI VERSION    : CRAY MPICH version 8.1.23.5 (ANL base 3.4a2)
MPI BUILD INFO : Tue Nov 29 13:44 2022 (git hash 210ae8b)

task    thread             node name  first task    # on node  core
total time      7.004
0000      0006         X1005C2S3B1N1        0000         0000  0006
0000      0005         X1005C2S3B1N1        0000         0000  0005
0000      0002         X1005C2S3B1N1        0000         0000  0002
0000      0000         X1005C2S3B1N1        0000         0000  0000
0000      0004         X1005C2S3B1N1        0000         0000  0004
0000      0003         X1005C2S3B1N1        0000         0000  0003
0000      0001         X1005C2S3B1N1        0000         0000  0001
0000      0007         X1005C2S3B1N1        0000         0000  0007
0001      0006         X1005C3S1B0N0        0001         0000  0006
0001      0005         X1005C3S1B0N0        0001         0000  0005
0001      0001         X1005C3S1B0N0        0001         0000  0001
0001      0004         X1005C3S1B0N0        0001         0000  0004
0001      0003         X1005C3S1B0N0        0001         0000  0003
0001      0000         X1005C3S1B0N0        0001         0000  0000
0001      0002         X1005C3S1B0N0        0001         0000  0002
0001      0007         X1005C3S1B0N0        0001         0000  0007
mpi_init 0 1696732969.0933 1696732969.2385       0.1452
mpi_init 1 1696732969.0089 1696732969.2390       0.2301
```

Example Output PrgEnv-gnu

```
MPI VERSION MPI VERSION    : CRAY MPICH version 8.1.23.5 (ANL base 3.4a2)
MPI BUILD INFO : Tue Nov 29 12:42 2022 (git hash 210ae8b)

task    thread             node name  first task    # on node  core
total time       7.004
0000      0001          X1005C2S3B1N1        0000          0000  0001
0000      0003          X1005C2S3B1N1        0000          0000  0003
0000      0005          X1005C2S3B1N1        0000          0000  0005
0000      0004          X1005C2S3B1N1        0000          0000  0004
0000      0006          X1005C2S3B1N1        0000          0000  0006
0000      0000          X1005C2S3B1N1        0000          0000  0000
0000      0002          X1005C2S3B1N1        0000          0000  0002
0000      0007          X1005C2S3B1N1        0000          0000  0007
0001      0000          X1005C3S1B0N0        0001          0000  0000
0001      0002          X1005C3S1B0N0        0001          0000  0002
0001      0001          X1005C3S1B0N0        0001          0000  0001
0001      0006          X1005C3S1B0N0        0001          0000  0006
0001      0004          X1005C3S1B0N0        0001          0000  0004
0001      0005          X1005C3S1B0N0        0001          0000  0005
0001      0007          X1005C3S1B0N0        0001          0000  0007
0001      0003          X1005C3S1B0N0        0001          0000  0003
mpi_init 0 1696732984.4138 1696732984.5317       0.1179
mpi_init 1 1696732984.3726 1696732984.5323       0.1597
```

Interactions between PrgvEnv-* and Intel

- Many "intel" modules most of which you can ignore
- Loading PrgvEnv-intel loads a particular Intel compiler module
- Should not try to load PrgvEnv-intel and Intel MPI at the same time
- For running Intel MPI there are only two choices of module load gcc that work.

Module	Contains					Who
intel-classic-mixed/2023.2.0	icc	icx	ifort	ifx	NO_mpicc	NREL
intel-oneapi-mixed/2023.2.0	icc	icx	ifort	ifx	NO_mpicc	NREL
intel-oneapi/2023.2.0	icc	icx	ifort	ifx	NO_mpicc	NREL
intel-classic/2023.2.0	icc	icx	ifort	ifx	NO_mpicc	NREL
intel-oneapi-compilers/2023.2.0	icc	icx	ifort	ifx	NO_mpicc	NREL
intel-oneapi-mpi/2021.10.0-intel	NO_icc	NO_icx	NO_ifort	NO_ifx	mpicc	NREL
intel/2023.2.0	icc	icx	ifort	ifx	NO_mpicc	NREL

```
Module Specific Help for "intel-oneapi-compilers/2023.2.0"
Name   : intel-oneapi-compilers
Version: 2023.2.0
Target : icelake

Intel oneAPI Compilers. Includes: icc, icpc, ifort, icx, icpx, ifx, and
dpcpp. LICENSE INFORMATION: By downloading and using this software, you
agree to the terms and conditions of the software license agreements at
https://intel.ly/393CijO.
```

```
Module Specific Help for "intel-oneapi-mpi/2021.10.0-intel"
Name   : intel-oneapi-mpi
Version: 2021.10.0
Target : icelake

Intel MPI Library is a multifabric message-passing library that
implements the open-source MPICH specification. Use the library to
create, maintain, and test advanced, complex applications that perform
better on high-performance computing (HPC) clusters based on Intel
processors. LICENSE INFORMATION: By downloading and using this software,
you agree to the terms and conditions of the software license agreements
at https://intel.ly/393CijO.
```

```
Module Specific Help for "intel-classic/2023.2.0" (and all of the others)
2023.2.0
/nopt/nrel/apps/compilers/08-23/spack/opt/spack/linux-rhel8-icelake/gcc-8.4.0/intel-oneapi-compilers-2023.2.0-
cqpelkddr7kvjjmbqgs5ypz27m2bgqgt/compiler/2023.2.0
This modulefile defines the system paths and environment variables needed to use
the Intel 'classic' compilers icc, ifort and icpc on Cray XE, XC, and XE
systems. This modulefile may be loadedi as a standalone or as part of
PrgEnv-intel, when the user will call these products using cc, ftn, and CC.
If loaded as part of PrgEnv-intel, it cannot be unloaded individually, but it
can be swapped for another version.
```

Loading "cray" versions of gcc after loading Intel can "kick out" Intel MPI

Can give you an unexpected mix of Intel and Cray software.

Module loaded with **module load intel-oneapi-mpi** **xxxxxxxx** **module load gcc/***	**Purge First**	**gcc (12.2.0)**	**gcc/10.1.0**	**gcc**/10.3.0	**gcc**/11.2.0	**gcc**/12.1.0	**gcc**/12.2.0	**gcc/13.1.0**
`intel-oneapi-compilers`	No	CrayMPI	IntelMPI	CrayMPI	CrayMPI	CrayMPI	CrayMPI	IntelMPI
`intel-oneapi-compilers`	Yes	IntelMPI	IntelMPI	IntelMPI	IntelMPI	IntelMPI	IntelMPI	IntelMPI
`intel-oneapi`	No	CrayMPI	IntelMPI	CrayMPI	CrayMPI	CrayMPI	CrayMPI	IntelMPI
`intel-oneapi`	Yes	IntelMPI	IntelMPI	IntelMPI	IntelMPI	IntelMPI	IntelMPI	IntelMPI

icc available / MPI version

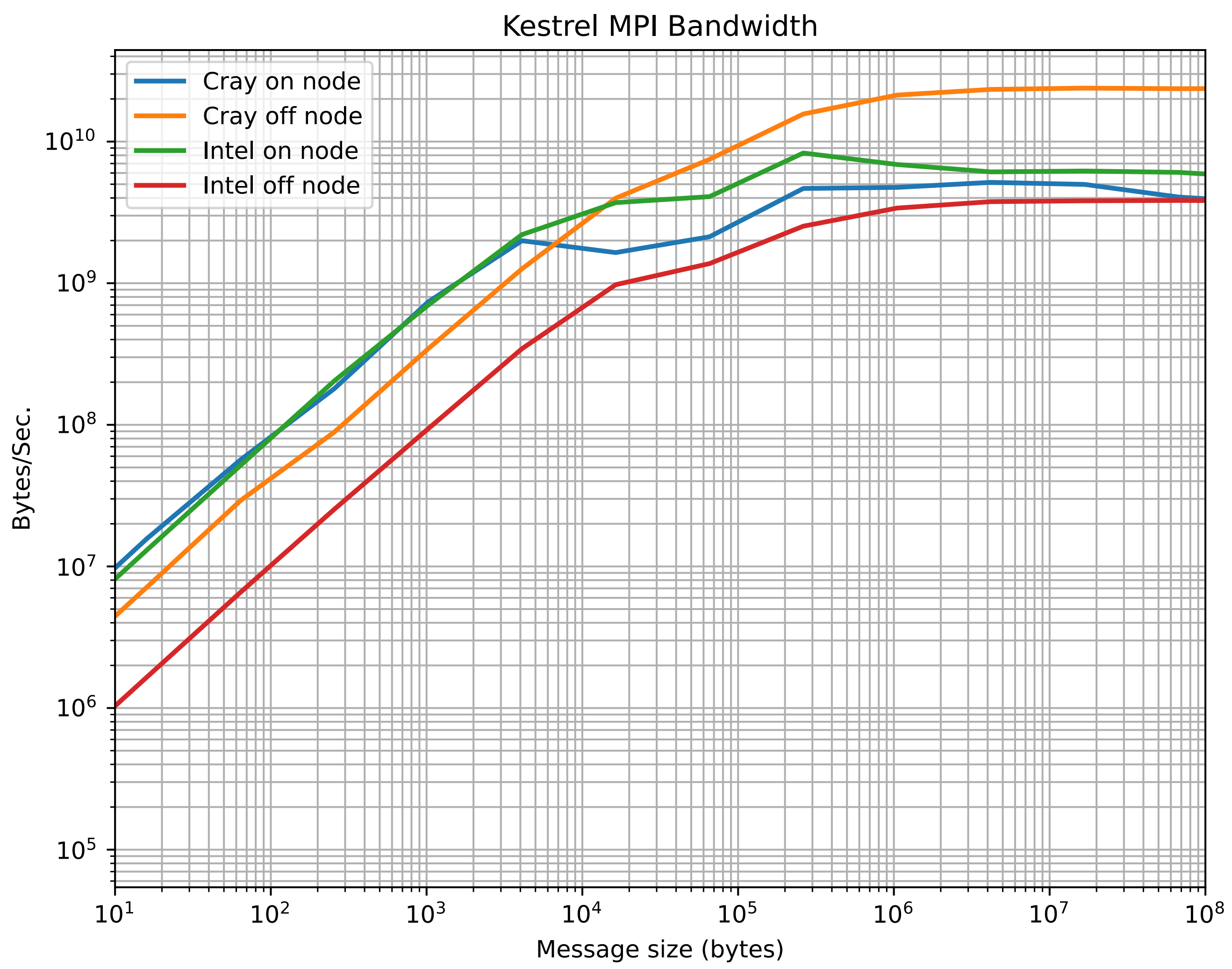
module load **cray-mpich-abi/8.1.23**

- module load cray-mpich-abi/8.1.23
  - Replaces Intel MPI with Cray MPI at runtime
  - Useful for cases where you have a binary but not the source
  - In theory, also works with programs built with MPICH
  - The command ldd can be used to see what version of MPI is being called

```
[tkaiser2@x1000c0s0b0n0 216704]$ldd c.impi | grep libmpi.so
  libmpi.so.12 => /nopt/nrel/apps/mpi/08-23/spack/opt/spack/linux-rhel8-icelake/intel-2021.10.0/intel-oneapi-
mpi-2021.10.0-7iolquprezbcmmeapg7rlcwfsd4r3rc7/mpi/2021.10.0/lib/release/libmpi.so.12 (0x00007f0c71d21000)

[tkaiser2@x1000c0s0b0n0 216704]$srun -n 2 ./c.impi -F
MPI VERSION Intel(R) MPI Library 2021.10 for Linux* OS

task    thread             node name  first task    # on node  core
0000      0000        x1000c0s0b0n0        0000         0000  0051
0000      0001        x1000c0s0b0n0        0000         0000  0039
0001      0000        x1000c0s0b0n0        0000         0001  0103
0001      0001        x1000c0s0b0n0        0000         0001  0091


[tkaiser2@x1000c0s0b0n0 216704]$module load cray-mpich-abi/8.1.23

Lmod is automatically replacing "cray-mpich/8.1.23" with "cray-mpich-abi/8.1.23".

[tkaiser2@x1000c0s0b0n0 216704]$ldd c.impi | grep libmpi.so
  libmpi.so.12 => /opt/cray/pe/mpich/8.1.23/ofi/crayclang/10.0/lib-abi-mpich/libmpi.so.12 (0x00007fae15df4000)

[tkaiser2@x1000c0s0b0n0 216704]$srun -n 2 ./c.impi -F
MPI VERSION MPI VERSION    : CRAY MPICH version 8.1.23.5 (ANL base 3.4a2)
MPI BUILD INFO : Tue Nov 29 12:42 2022 (git hash 210ae8b)

task    thread             node name  first task    # on node  core
0000      0000        x1000c0s0b0n0        0000         0000  0051
0000      0001        x1000c0s0b0n0        0000         0000  0039
0001      0000        x1000c0s0b0n0        0000         0001  0103
0001      0001        x1000c0s0b0n0        0000         0001  0093
[tkaiser2@x1000c0s0b0n0 216704]$
```

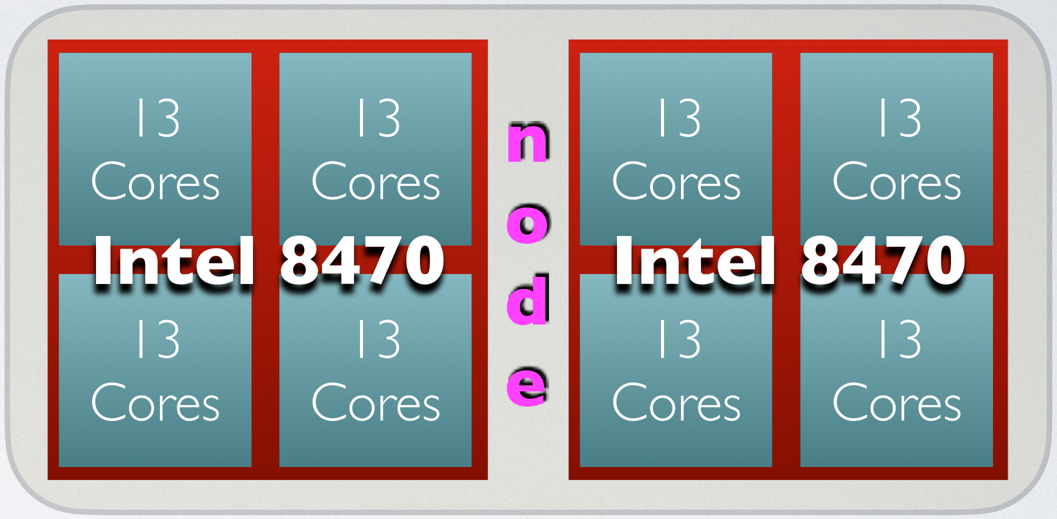
BACK TO PINGPONG

- Measures message speed as a function of size
- Measure on node and off node
- All PrgEnv-* will run at the same speed
- We compare Intel-MPI to PrgEnv-*
- Which is better:
    - Intel in general better on node
    - PrgEnv-* is better off node


Kestrel MPI Bandwidth
Cray on node
Cray off node
Intel on node
Intel off node
Bytes/Sec.
Message size (bytes)
10^{5}
10^{6}
10^{7}
10^{8}
10^{9}
10^{10}
10^{1}
10^{2}
10^{3}
10^{4}
10^{5}
10^{6}
10^{7}
10^{8}

AFFINITY & WHY IMPORTANT

- Affinity - mapping of threads/tasks to cores
- Kestrel 104 cores/node
  - 2 chips (Intel 8470)
    - 52 cores each
    - 4 "tiles" with 13 cores each



  - Worst case: Multiple threads/tasks can end up on the same core potentially reducing performance by 2X or maybe much more
  - Also: You may want to put threads/tasks on particular tiles to maximize communications or memory access
  - Possible to have different MPI tasks to have different # threads (Ask if interested)

CUT TO THE CHASE:

- With the proper setting in sbatch scripts and the srun command we are able to “trivially” get apps to behave reasonably for all cases I tested on TDS, Swift and Eagle.
- Masks (a mapping list) allow a fine grain placement of tasks & threads to cores

SUFFICIENT EXAMPLE

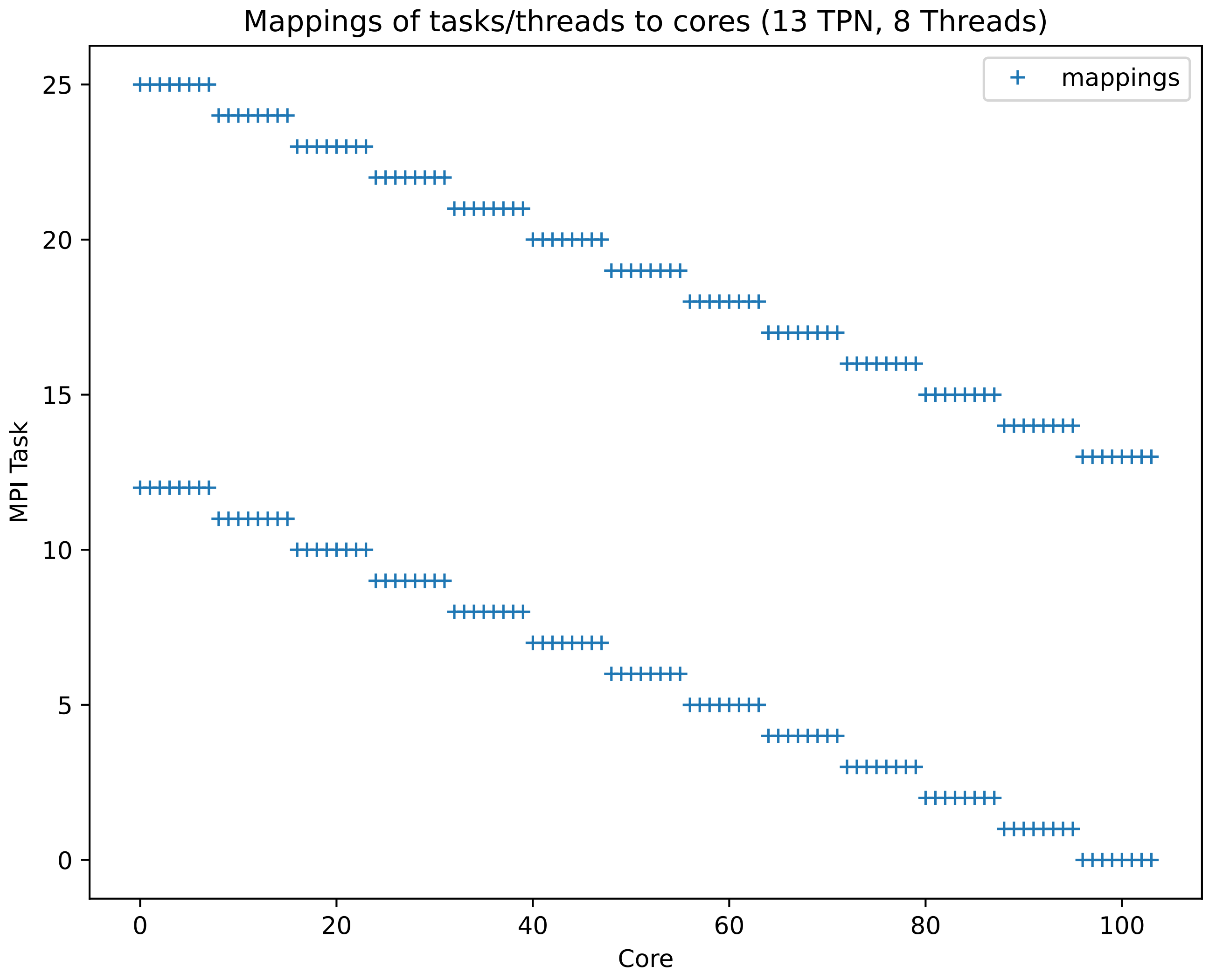
For 2 nodes, 18 tasks per node, 2 threads per task

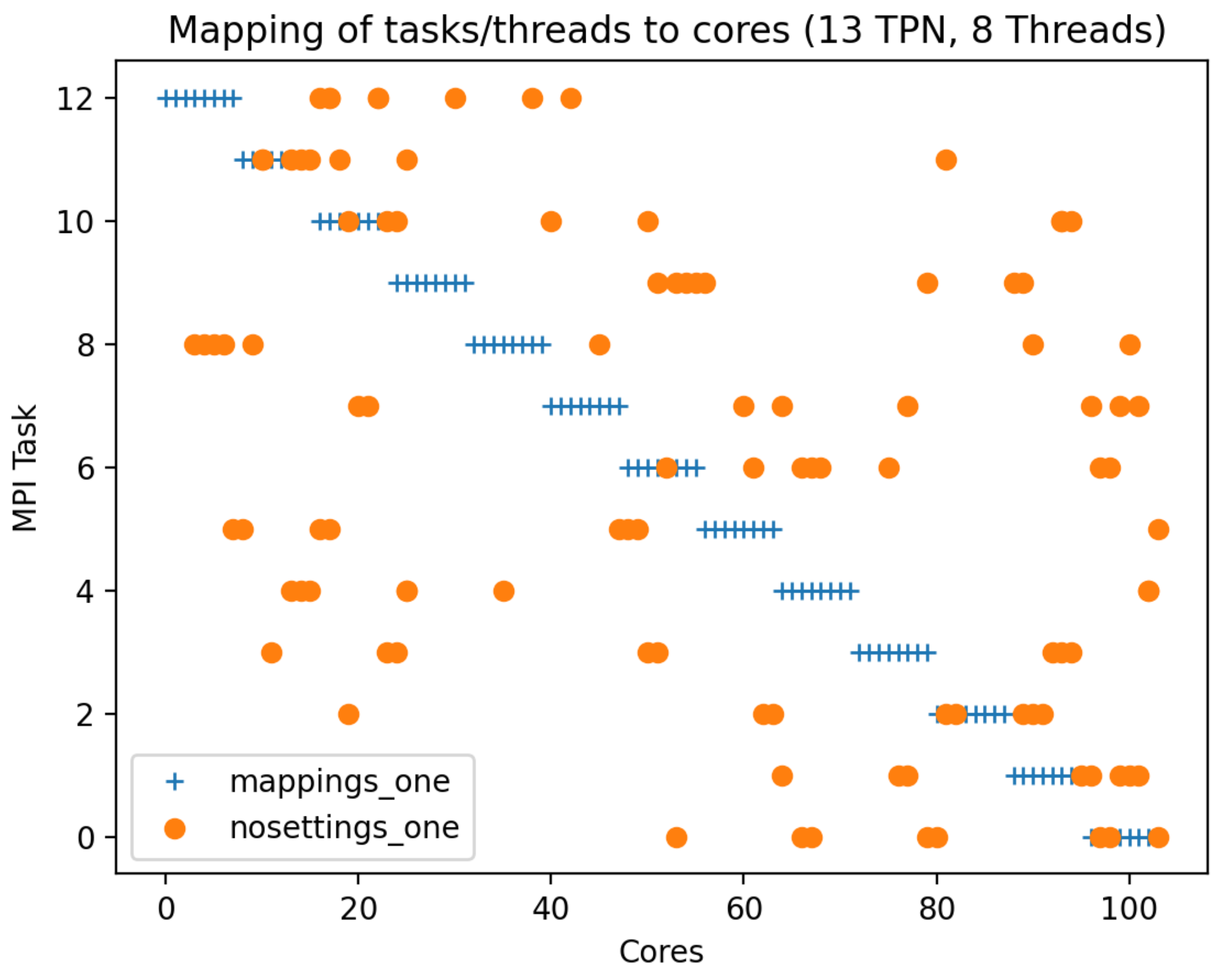
```
#!/usr/bin/bash
#SBATCH --nodes=2
#SBATCH --exclusive
#SBATCH --export=ALL


  export OMP_PLACES=cores
  export OMP_PROC_BIND=spread
  export OMP_NUM__THREADS=13

  srun --mpi=pmi2 --threads-per-core=1  --tasks-per-node=4 —cpus-per-task=13 ...
```

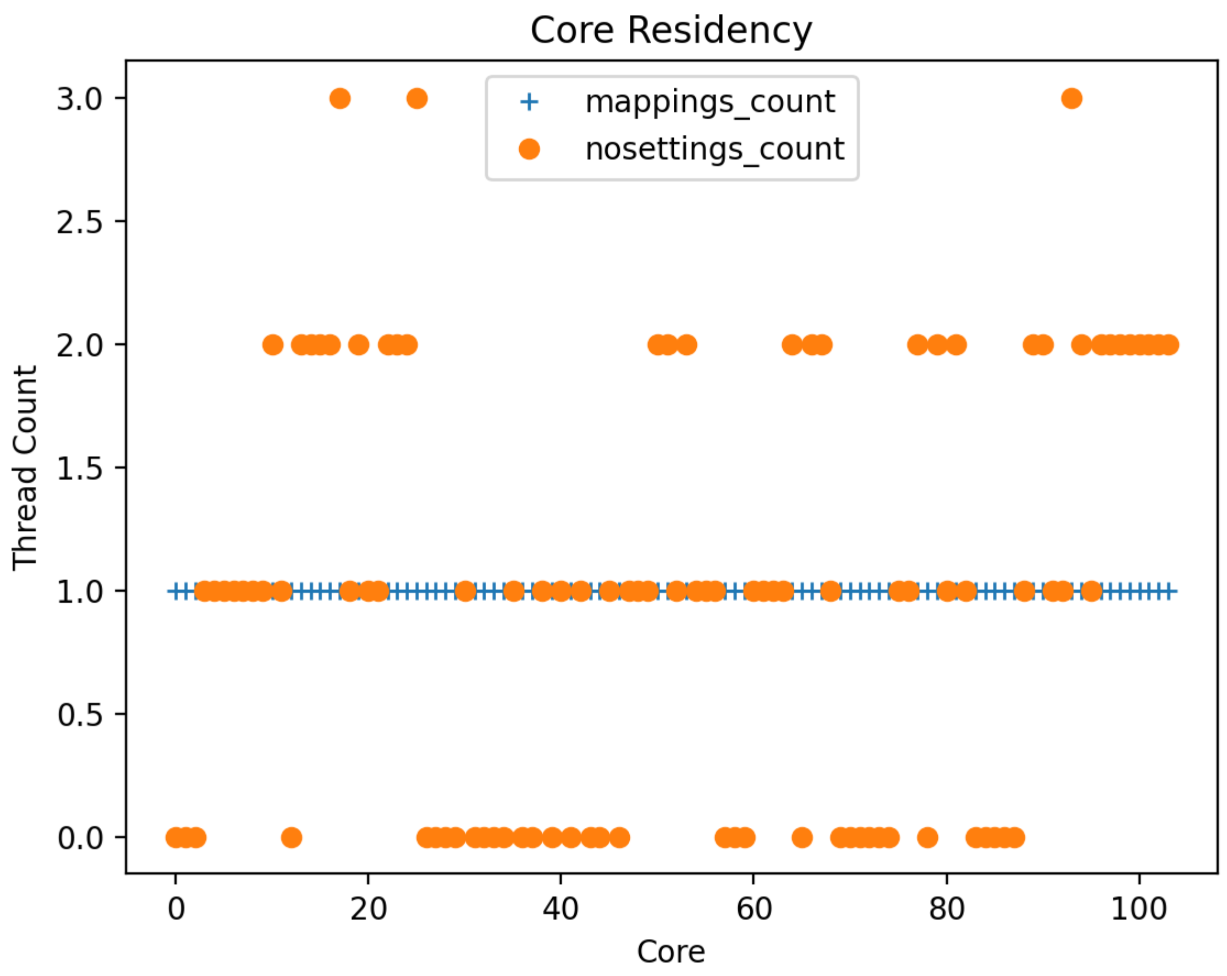
With the setting show in red we get good mappings for Kestrel, Eagle and Swift

--cpus-per-task = OMP_NUM_THREADS

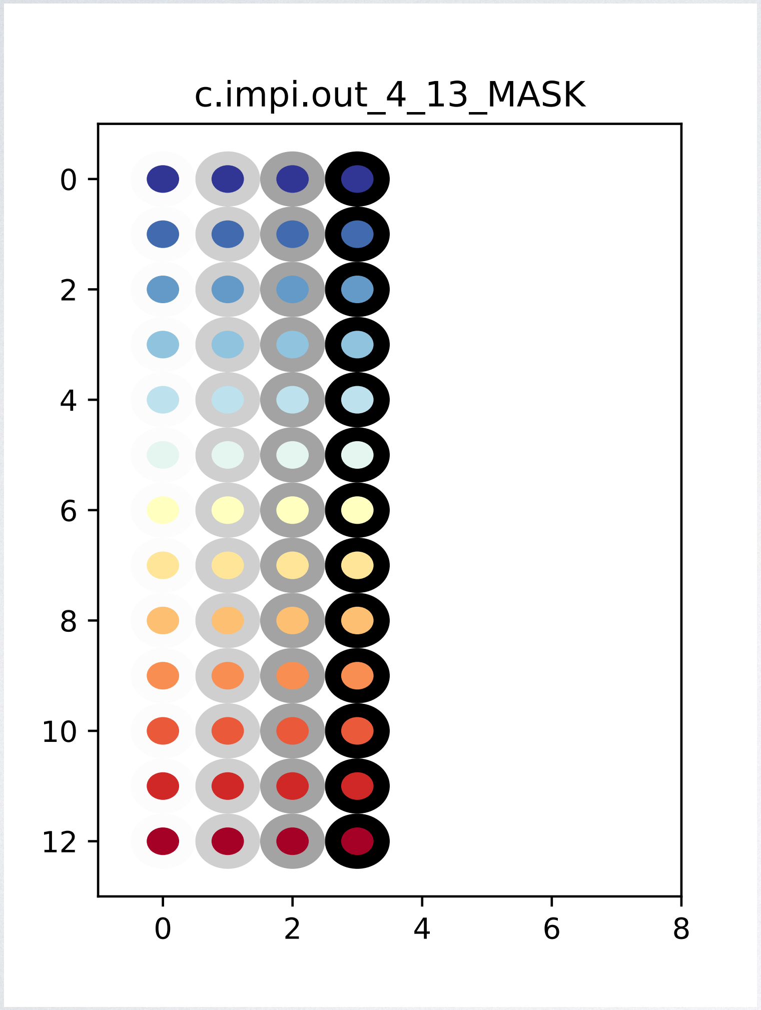
Add -cpu-bind=v to see the binding

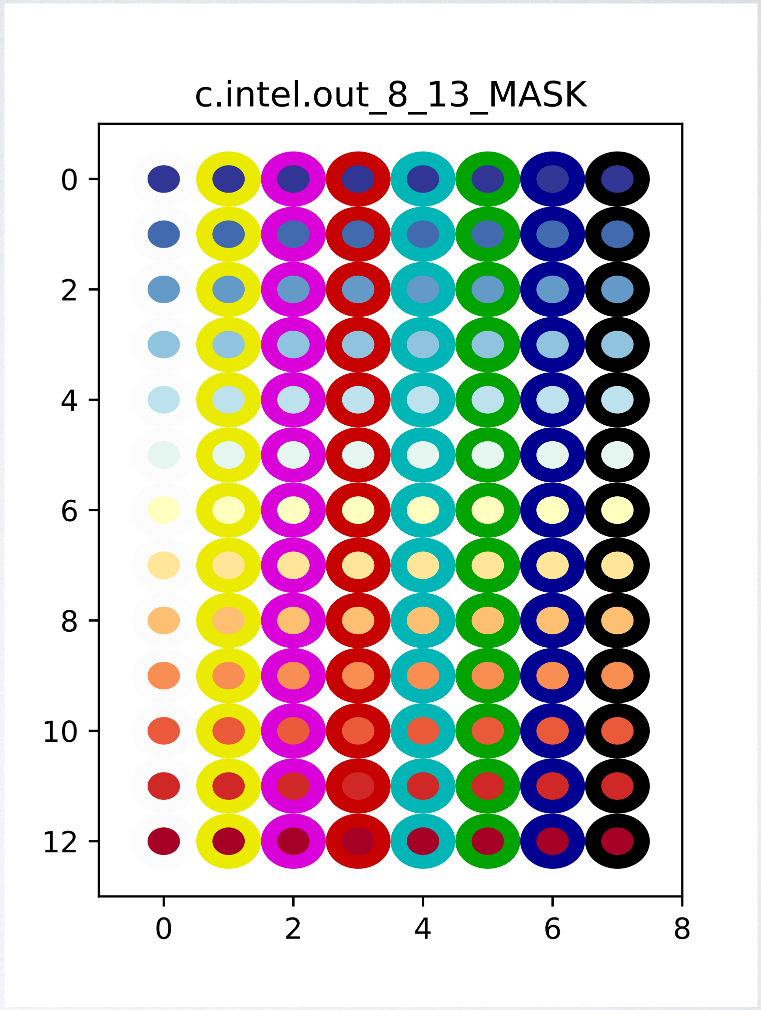

Mappings of tasks/threads to cores (13 TPN, 8 Threads)
mappings
MPI Task
Core
0
5
10
15
20
25
0
20
40
60
80
100


Mapping of tasks/threads to cores (13 TPN, 8 Threads)
MPI Task
Cores
0
2
4
6
8
10
12
0
20
40
60
80
100
mappings_one
nosettings_one


Core Residency
Thread Count
Core
mappings_count
nosettings_count
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0
20
40
60
80
100

EXAMPLE 4 & 8 TASKS/NODE 13 THREADS





Tests run

Tasks/node	Threads/task	Cores/node
104	1	104
52	2	104
52	1	52
26	4	104
26	2	52
26	1	26
13	4	52
13	2	26
13	1	13
8	13	104
4	26	104
4	13	52
4	8	32
2	52	104
2	26	52
2	13	26
2	8	16
1	104	104
1	91	91
1	78	78
1	65	65
1	52	52
1	26	26
1	13	13
1	8	8

MPI/Compiler sets

- PrgEnv-cray
- PrgEnv-gnu
- PrgEnv-intel
- intel-oneapi

OUR EXAMPLE SCRIPT

- Originally designed as an compile/run and affinity tester
- Sets up a new directory and goes there
- Copies all required files
- Does a make for all versions
- Loops over # MPI tasks and # OpenMP threads (input file cases)
    - Loops over two types of thread binding (scattered and manual)
        - Steps through MPI versions with both C and Fortran

OUR EXAMPLE SCRIPT

- Reports lots of information
  - Bindings for each run
  - Normal program output for phostrun includes mapping of tasks and threads to nodes and cores
  - MPI launch times
- Runs a single instance of ppong for each version of MPI (runs the setup script todo.py to create an input file)
- Final output is a report of successful/failed mapping
  - Success = expected unique combinations of nodes and cores
  - Good News: It works for all tested versions of MPI and mappings

OVERKILL FOR MOST PEOPLE

- While you can run this for the full set you might not want to use the allocation hours (minutes)
- Suggested use...
  - Run phostone using the exact run arguments you use for your production code to see how it maps tasks and threads to cores

```
#!/usr/bin/bash
#SBATCH --job-name="affinity"
#SBATCH --nodes=2
#SBATCH --exclusive
#SBATCH --export=ALL
#SBATCH --time=04:00:00
#SBATCH --partition=standard

BASE=`pwd`

#Make a new directory and go there
STDIR=`pwd`
mkdir $SLURM_JOB_ID
cd $SLURM_JOB_ID

#optionally wait between launches
mywait () { sleep 0; }

#Copy everything
printenv > env
cat $0 > script

cp $BASE/make* .
cp $BASE/Makefile .
cp $BASE/fhostone.F90 .
cp $BASE/phostone.c .
cp $BASE/cases .
cp $BASE/post .
cp $BASE/ppong.c .
cp $BASE/getcore.c .
cp $BASE/maskgenerator.py .
cp $BASE/todo.py .
cp $BASE/tymer  .

tar -czf recreate.tgz *

#Create input for ppong
./todo.py

#Build our programs
make all > make.log 2>&1
make pp  > make.pp 2>&1

#Command line arguments for phostone
CLA="-i -F -E -t 7"
export FEXE=f
export CEXE=c
```

Setup and "make"

make*	Makefiles
Makefile	Driver Makefile
cases	File of tasks and threads
post	Post processing script
maskgenerator.py	Manually creates mapping of threads to cores
todo.py	Creates input file for ppong
tymer	Nice wall clock timer

Makefile (full)

```
all : impi  cray  gnu  intel open mpich openg mpichg dmod

impi: makeimpi
      make -f makeimpi

cray: makeprgcray
      make -f makeprgcray

gnu: makeprggnu
      make -f makeprggnu

intel: makeprgintel
      make -f makeprgintel

open: makeopen
      make -f makeopen

mpich: makempich
      make -f makempich


openg: makeopen_g
      make -f makeopen_g

mpichg: makempich_g
      make -f makempich_g

clean:
      make -f makeimpi clean
      make -f makeprgintel clean
      make -f makeprggnu clean
      make -f makeprgcray clean
      make -f makeopen clean
      make -f makempich clean
      make -f makeopen_g clean
      make -f makempich_g clean
      rm -rf runall.tgz simple.tgz
```

```
dmod:
      rm -rf *.o *mod

pp: pp.impi pp.cray pp.gnu pp.intel pp.open pp.oneapi pp.mpich pp.openg pp.mpichg

pp.impi: makeimpi
      make -f makeimpi pp.impi

pp.cray: makeprgcray
      make -f makeprgcray pp.cray

pp.gnu: makeprggnu
      make -f makeprggnu pp.gnu

pp.intel: makeprgintel
      make -f makeprgintel pp.intel

pp.open: makeopen
      make -f makeopen pp.open

pp.mpich: makempich
      make -f makempich pp.mpich


pp.openg: makeopen_g
      make -f makeopen_g pp.openg

pp.mpichg: makempich_g
      make -f makempich_g pp.mpichg

tar:
      tar -czf runall.tgz \
            cases eagle ecases fhostone.F90 getcore.c make1api Makefile makefile.include \
            makeimpi makeopen makeprgcray makeprggnu makeprgintel maskgenerator.py masks.txt \
            phostone.c post ppong.c readme.md runall runpp subsweep sweep todo.py tymer \
            scases array mapping.py simple makempich makempich_g makeopen_g

simple.tgz:
      tar -czf simple.tgz fhostone.F90  getcore.c  make1api  Makefile  makefile.include  \
           makefile.org  makeimpi  makeopen  makeprgcray  makeprggnu  makeprgintel \
           phostone.c  post  ppong.c simple makempich makempich_g makeopen_g
```

Looping

- Get a task/thread count from each line of cases
- We try two types of thread binding, spread (NONE) and manually (MASK)

```
#LOOPING
export CRAY_OMP_CHECK_AFFINITY=TRUE
export nc=`cat cases | wc -l`
for il in `seq $nc` ; do
    aline=`cat cases | head -$il | tail -1`
    ntpn=`echo $aline | awk {'print $1'}`
    nthrd=`echo $aline | awk {'print $2'}`
    export OMP_NUM_THREADS=$nthrd
    for bindit in NONE MASK ; do
        #export KMP_AFFINITY=scatter
        export OMP_PROC_BIND=spread
        export BIND=--cpu-bind=v,${bindit}
        unset CPUS_TASK
        if [ $bindit == MASK ] ; then
        cores=`expr $ntpn \* $nthrd`
        MASK=`./maskgenerator.py $cores $ntpn`
        BIND="--cpu-bind=v,mask_cpu:$MASK"
        fi
        if [ $bindit == NONE ] ; then
        BIND="--cpu-bind=v"
          export CPUS_TASK="--cpus-per-task=$nthrd"
        fi
        echo $ntpn $nthrd >> srunsettings
        echo $BIND $CPUS_TASK >> srunsettings
        printenv | egrep "OMP_|KMP_" >> srunsettings
        echo --mpi=pmi2 $BIND  --tasks-per-node=$ntpn $CPUS_TASK >> srunsettings
...
...
```

- The script maskgenerator.py creates a string describing a mapping of tasks/threads to cores
- This is passed to run using the --cpu-bind option

- Save information for each iteration

```
./tymer mytimes PrgEnv-intel
          module purge
          module load craype-x86-spr
          module load intel
          module load PrgEnv-intel

./tymer mytimes fortran
          mywait; srun --mpi=pmi2 $BIND  --tasks-per-node=$ntpn $CPUS_TASK  ./$FEXE.intel $CLA > f.intel.out_${ntpn}_${nthrd}_${bindit} \
                 2> f.intel.info_${ntpn}_${nthrd}_${bindit}
./tymer mytimes c
          mywait; srun --mpi=pmi2 $BIND  --tasks-per-node=$ntpn $CPUS_TASK  ./$CEXE.intel $CLA > c.intel.out_${ntpn}_${nthrd}_${bindit} \
                 2> c.intel.info_${ntpn}_${nthrd}_${bindit}
./tymer mytimes finished

          if [[ $nthrd -eq 1 &&  $ntpn -eq 104 && $bindit == NONE ]] ; then
          mywait; srun --mpi=pmi2 $BIND  --tasks-per-node=$ntpn $CPUS_TASK  ./pp.intel $CLA > pp.intel.xxx_${ntpn}_${nthrd}_${bindit} \
                                   2> pp.intel.iii_${ntpn}_${nthrd}_${bindit}
./tymer mytimes finished ppong
          fi

./tymer mytimes PrgEnv-gnu
          module purge
          module load craype-x86-spr
          module load PrgEnv-gnu

./tymer mytimes fortran
          mywait; srun --mpi=pmi2 $BIND  --tasks-per-node=$ntpn $CPUS_TASK  ./$FEXE.gnu $CLA > f.gnu.out_${ntpn}_${nthrd}_${bindit} \
                 2> f.gnu.info_${ntpn}_${nthrd}_${bindit}
./tymer mytimes c
          mywait; srun --mpi=pmi2 $BIND  --tasks-per-node=$ntpn $CPUS_TASK  ./$CEXE.gnu $CLA > c.gnu.out_${ntpn}_${nthrd}_${bindit} \
                 2> c.gnu.info_${ntpn}_${nthrd}_${bindit}
./tymer mytimes finished

          if [[ $nthrd -eq 1 &&  $ntpn -eq 104 && $bindit == NONE ]] ; then
          mywait; srun --mpi=pmi2 $BIND  --tasks-per-node=$ntpn $CPUS_TASK  ./pp.gnu $CLA > pp.gnu.xxx_${ntpn}_${nthrd}_${bindit} \
                                   2> pp.gnu.iii_${ntpn}_${nthrd}_${bindit}
./tymer mytimes finished ppong
          fi
```

```
./tymer mytimes PrgEnv-cray
    module purge
    module load craype-x86-spr
    module load PrgEnv-cray

./tymer mytimes fortran
    mywait; srun --mpi=pmi2 $BIND  --tasks-per-node=$ntpn $CPUS_TASK  ./$FEXE.cray $CLA > f.cray.out_${ntpn}_${nthrd}_${bindit} \
        2> f.cray.info_${ntpn}_${nthrd}_${bindit}
./tymer mytimes c
    mywait; srun --mpi=pmi2 $BIND  --tasks-per-node=$ntpn $CPUS_TASK  ./$CEXE.cray $CLA > c.cray.out_${ntpn}_${nthrd}_${bindit} \
        2> c.cray.info_${ntpn}_${nthrd}_${bindit}
./tymer mytimes finished

    if [[ $nthrd -eq 1 &&  $ntpn -eq 104 && $bindit == NONE ]] ; then
    mywait; srun --mpi=pmi2 $BIND  --tasks-per-node=$ntpn $CPUS_TASK  ./pp.cray $CLA > pp.cray.xxx_${ntpn}_${nthrd}_${bindit} \
              2> pp.cray.iii_${ntpn}_${nthrd}_${bindit}
./tymer mytimes finished ppong
    fi

./tymer mytimes intel-oneapi
    module purge
    module load intel-oneapi
    module load libfabric

./tymer mytimes fortran
    mywait; srun --mpi=pmi2 $BIND  --tasks-per-node=$ntpn $CPUS_TASK  ./$FEXE.impi $CLA > f.impi.out_${ntpn}_${nthrd}_${bindit} \
        2> f.impi.info_${ntpn}_${nthrd}_${bindit}
./tymer mytimes c
    mywait; srun --mpi=pmi2 $BIND  --tasks-per-node=$ntpn $CPUS_TASK  ./$CEXE.impi $CLA > c.impi.out_${ntpn}_${nthrd}_${bindit} \
        2> c.impi.info_${ntpn}_${nthrd}_${bindit}
./tymer mytimes finished

    if [[ $nthrd -eq 1 &&  $ntpn -eq 104 && $bindit == NONE ]] ; then
    mywait; srun --mpi=pmi2 $BIND  --tasks-per-node=$ntpn $CPUS_TASK  ./pp.impi $CLA > pp.impi.xxx_${ntpn}_${nthrd}_${bindit} \
              2> pp.impi.iii_${ntpn}_${nthrd}_${bindit}
./tymer mytimes finished ppong
    fi
    done
done
```

Post processing

- Report of successful and failed phostone runs
- Copy slurm output to our final directory
- Might want to look at output from ppong
  - Bandwidth
  - MPI_Barrier rate
- Might want to look at MPI_Init times from phostone

```
./post
. ./post | sort -n > posit
getstate postit nope > report
getstate postit worked >> report
mv $STDIR/slurm-$SLURM_JOB_ID.out .
```

REPORT

```
tkaiser2-37907s:177402 tkaiser2$ cat report
c.cray.out_104_1_MASK 208 208
c.cray.out_104_1_NONE 208 208
c.cray.out_1_104_MASK 208 208
c.cray.out_1_104_NONE 208 208
c.cray.out_1_8_MASK 16 16
c.cray.out_1_8_NONE 16 16
c.gnu.out_104_1_MASK 208 208
c.gnu.out_104_1_NONE 208 208
c.gnu.out_1_104_MASK 208 208
c.gnu.out_1_104_NONE 208 208
c.gnu.out_1_8_MASK 16 16
c.gnu.out_1_8_NONE 16 16
c.impi.out_104_1_MASK 208 208
c.impi.out_104_1_NONE 208 208
c.impi.out_1_104_MASK 208 208
c.impi.out_1_104_NONE 208 208
c.impi.out_1_8_MASK 16 16
c.impi.out_1_8_NONE 16 16
c.intel.out_104_1_MASK 208 208
c.intel.out_104_1_NONE 208 208
c.intel.out_1_104_MASK 208 208
c.intel.out_1_104_NONE 208 208
c.intel.out_1_8_MASK 16 16
c.intel.out_1_8_NONE 16 16
```

```
f.cray.out_104_1_MASK 208 208
f.cray.out_104_1_NONE 208 208
f.cray.out_1_104_MASK 208 208
f.cray.out_1_104_NONE 208 208
f.cray.out_1_8_MASK 16 16
f.cray.out_1_8_NONE 16 16
f.gnu.out_104_1_MASK 208 208
f.gnu.out_104_1_NONE 208 208
f.gnu.out_1_104_MASK 208 208
f.gnu.out_1_104_NONE 208 208
f.gnu.out_1_8_MASK 16 16
f.gnu.out_1_8_NONE 16 16
f.impi.out_104_1_MASK 208 208
f.impi.out_104_1_NONE 208 208
f.impi.out_1_104_MASK 208 208
f.impi.out_1_104_NONE 208 208
f.impi.out_1_8_MASK 16 16
f.impi.out_1_8_NONE 16 16
f.intel.out_104_1_MASK 208 208
f.intel.out_104_1_NONE 208 208
f.intel.out_1_104_MASK 208 208
f.intel.out_1_104_NONE 208 208
f.intel.out_1_8_MASK 16 16
f.intel.out_1_8_NONE 16 16
```

WE ARE SKIPPING SOME INSTALLED VERSIONS OF MPI

- Skipped
```
openmpi/4.1.5-gcc
openmpi/4.1.5-intel
mpich/4.1-gcc
mpich/4.1-intel
```
- Don't perform as well
- Running a single instance of these usually works.
- Running many is succession has a bad habit of hanging

mpich/4.1-gcc

```
SHELL:=/usr/bin/bash

recurse:
	module purge                       ; \
	module load mpich/4.1-gcc   ; \
	module load gcc/13.1.0        ;\
	$(MAKE) -f $(firstword $(MAKEFILE_LIST)) both

# To use intelversion of the compilers
# replace thw two lines above with these
#  module load mpich/4.1-intel ; \
#  module load intel-oneapi ;\
# You should replace the same lines in the run script.

both: f.mpichg c.mpichg pp.mpichg


#defines USEFAST
include makefile.include

ifeq ($(USEFAST),yes)
#OPS=-DUSEFAST
#EXTRA=getcore.o
endif

F90=mpif90
CC=mpicc -lm

f.mpichg: fhostone.F90 $(EXTRA)
	$(F90) $(OPS) $(EXTRA) -fopenmp fhostone.F90 -O3 -o f.mpichg
	rm -f getcore.o

c.mpichg: phostone.c
	$(CC) $(OPS) -fopenmp phostone.c -O3 -o c.mpichg

pp.mpichg: ppong.c
	$(CC) $(OPS) $(WES) ppong.c -O3 -o pp.mpichg

clean:
	rm -rf *o *mod* f.mpichg c.mpichg pp.mpichg
```

mpich/4.1-intel

```
SHELL:=/usr/bin/bash

recurse:
	module purge                      ; \
	module load mpich/4.1-intel       ; \
	module load intel-oneapi          ; \
	$(MAKE) -f $(firstword $(MAKEFILE_LIST)) both

# To use gcc version of the compilers
# replace thw two lines above with these
#  module load mpich/4.1-gcc   ; \
#  module load gcc        ;\
# You should repleace the same lines in the run script.

both: f.mpich c.mpich pp.mpich


#defines USEFAST
include makefile.include

ifeq ($(USEFAST),yes)
#OPS=-DUSEFAST
#EXTRA=getcore.o
endif

F90=mpif90
CC=mpicc -lm

f.mpich: fhostone.F90 $(EXTRA)
	$(F90) $(OPS) $(EXTRA) -fopenmp fhostone.F90 -O3 -o f.mpich
	rm -f getcore.o

c.mpich: phostone.c
	$(CC) $(OPS) -fopenmp phostone.c -O3 -o c.mpich

pp.mpich: ppong.c
	$(CC) $(OPS) $(WES) ppong.c -O3 -o pp.mpich

clean:
	rm -rf *o *mod* f.mpich c.mpich pp.mpich
```

openmpi/4.1.5-intel

```
SHELL:=/usr/bin/bash

recurse:
	module purge                       ; \
	module load  openmpi/4.1.5-intel   ; \
	module load intel-oneapi           ; \
	$(MAKE) -f $(firstword $(MAKEFILE_LIST)) both

# To run with gcc /Openmpi replace the lines above with these
#  module load openmpi/4.1.5-gcc ; \
#  module load gcc    ; \
# You should also replace the lines in the run script.

both: f.open c.open pp.open

#defines USEFAST
include makefile.include

ifeq ($(USEFAST),yes)
#OPS=-DUSEFAST
#EXTRA=getcore.o
endif

F90=mpif90
CC=mpicc -lm

f.open: fhostone.F90 $(EXTRA)
	$(F90) $(OPS) $(EXTRA) -fopenmp fhostone.F90 -O3 -o f.open
	rm -f getcore.o

c.open: phostone.c
	$(CC) $(OPS) -fopenmp phostone.c -O3 -o c.open

pp.open: ppong.c
	$(CC) $(OPS) $(WES) ppong.c -O3 -o pp.open

clean:
	rm -rf *o *mod* f.open c.open pp.open
```

openmpi/4.1.5-gcc

```
SHELL:=/usr/bin/bash

recurse:
	module purge                       ; \
	module load  openmpi/4.1.5-gcc     ; \
	module load gcc/13.1.0             ; \
	$(MAKE) -f $(firstword $(MAKEFILE_LIST)) both

# To run with intel / Openmpi replace the lines above with these
#  module load openmpi/4.1.5-intel ; \
#  module load intel-oneapi   ; \
# You should also replace the lines in the run script.

both: f.openg c.openg pp.openg

#defines USEFAST
include makefile.include

ifeq ($(USEFAST),yes)
#OPS=-DUSEFAST
#EXTRA=getcore.o
endif

F90=mpif90
CC=mpicc -lm

f.openg: fhostone.F90 $(EXTRA)
	$(F90) $(OPS) $(EXTRA) -fopenmp fhostone.F90 -O3 -o f.openg
	rm -f getcore.o

c.openg: phostone.c
	$(CC) $(OPS) -fopenmp phostone.c -O3 -o c.openg

pp.openg: ppong.c
	$(CC) $(OPS) $(WES) ppong.c -O3 -o pp.openg

clean:
	rm -rf *o *mod* f.openg c.openg pp.openg
```

```
:<<SKIP
           tymer mytimes openmpi/4.1.5-gcc
           module purge
           module load openmpi/4.1.5-gcc
           module load gcc
           mywait; srun --mpi=pmi2 $BIND  --tasks-per-node=$ntpn $CPUS_TASK  ./$FEXE.openg $CLA > f.openg.out_${ntpn}_${nthrd}_${bindit} \
                        2> f.openg.info_${ntpn}_${nthrd}_${bindit}
           mywait; srun --mpi=pmi2 $BIND  --tasks-per-node=$ntpn $CPUS_TASK  ./$CEXE.openg $CLA > c.openg.out_${ntpn}_${nthrd}_${bindit} \
                        2> c.openg.info_${ntpn}_${nthrd}_${bindit}
./tymer mytimes finished

           if [[ $nthrd -eq 1 &&  $ntpn -eq 104 && $bindit == NONE ]] ; then
           mywait; srun --mpi=pmi2 $BIND  --tasks-per-node=$ntpn $CPUS_TASK  ./ppong.openg $CLA > pp.openg.xxx_${ntpn}_${nthrd}_${bindit}
                                        2> pp.openg.iii_${ntpn}_${nthrd}_${bindit}
./tymer mytimes finished ppong
           fi

           tymer mytimes openmpi/4.1.5-intel
           module purge
           module load openmpi/4.1.5-intel
           module load intel-oneapi
           mywait; srun --mpi=pmi2 $BIND  --tasks-per-node=$ntpn $CPUS_TASK  ./$FEXE.open $CLA > f.open.out_${ntpn}_${nthrd}_${bindit} \
                        2> f.open.info_${ntpn}_${nthrd}_${bindit}
           mywait; srun --mpi=pmi2 $BIND  --tasks-per-node=$ntpn $CPUS_TASK  ./$CEXE.open $CLA > c.open.out_${ntpn}_${nthrd}_${bindit} \
                        2> c.open.info_${ntpn}_${nthrd}_${bindit}

./tymer mytimes finished

           if [[ $nthrd -eq 1 &&  $ntpn -eq 104 && $bindit == NONE ]] ; then
           mywait; srun --mpi=pmi2 $BIND  --tasks-per-node=$ntpn $CPUS_TASK  ./ppong.open $CLA > pp.open.xxx_${ntpn}_${nthrd}_${bindit}
                                        2> pp.iopen.iii_${ntpn}_${nthrd}_${bindit}
```

```
./tymer mytimes finished ppong
          fi
          tymer mytimes mpich/4.1-intel
          module purge
          module load mpich/4.1-intel
          module load intel-oneapi
          module load libfabric
          unset UCX_NET_DEVICES
          mywait; srun --mpi=pmi2 $BIND  --time=00:03:00 --tasks-per-node=$ntpn $CPUS_TASK  ./$FEXE.mpich $CLA > f.mpich.out_${ntpn}_${nthrd}_${bindit} \
                      2> f.mpich.info_${ntpn}_${nthrd}_${bindit}
          mywait; srun --mpi=pmi2 $BIND  --time=00:03:00 --tasks-per-node=$ntpn $CPUS_TASK  ./$CEXE.mpich $CLA > c.mpich.out_${ntpn}_${nthrd}_${bindit} \
                      2> c.mpich.info_${ntpn}_${nthrd}_${bindit}
./tymer mytimes finished

          if [[ $nthrd -eq 1 &&  $ntpn -eq 104 && $bindit == NONE ]] ; then
          mywait; srun --mpi=pmi2 $BIND  --tasks-per-node=$ntpn $CPUS_TASK  ./ppong.mpich $CLA > pp.mpich.xxx_${ntpn}_${nthrd}_${bindit}
                                        2> pp.mpich.iii_${ntpn}_${nthrd}_${bindit}
./tymer mytimes finished ppong
          fi

          tymer mytimes mpich/4.1-gcc
          module purge
          module load mpich/4.1-gcc
          module load gcc
          module load libfabric
          unset UCX_NET_DEVICES
          mywait; srun --mpi=pmi2 $BIND  --time=00:03:00 --tasks-per-node=$ntpn $CPUS_TASK  ./$FEXE.mpichg $CLA > f.mpichg.out_${ntpn}_${nthrd}_${bindit} \
                      2> f.mpichg.info_${ntpn}_${nthrd}_${bindit}
          mywait; srun --mpi=pmi2 $BIND  --time=00:03:00 --tasks-per-node=$ntpn $CPUS_TASK  ./$CEXE.mpichg $CLA > c.mpichg.out_${ntpn}_${nthrd}_${bindit} \
                      2> c.mpichg.info_${ntpn}_${nthrd}_${bindit}
./tymer mytimes finished

          if [[ $nthrd -eq 1 &&  $ntpn -eq 104 && $bindit == NONE ]] ; then
          mywait; srun --mpi=pmi2 $BIND  --tasks-per-node=$ntpn $CPUS_TASK  ./ppong.mpichg $CLA > pp.mpichg.xxx_${ntpn}_${nthrd}_${bindit}
                                        2> pp.mpichg.iii_${ntpn}_${nthrd}_${bindit}
./tymer mytimes finished ppong
          fi
SKIP
```